**rpc** **getMySubscriptions(SubscriptionsRequest) returns (SubscriptionsResponse);**

**~~rpc getSessions(IntervalsRequest) returns (IntervalsResponse);~~**

**rpc addNewUser(NewUserRequest) returns (NewUserResponse);**

~~rpc updateGCMToken(UpdateTokenRequest) returns (UpdateTokenResponse);~~

**rpc addNewSubscription(NewSubscriptionRequest) returns (NewSubscriptionResponse);**

**rpc** **getSubscriptionNumberState(SubscriptionNumberStateRequest) returns (SubscriptionNumberStateResponse);**

rpc getAvatar(AvatarRequest) returns (AvatarResponse);

**~~rpc unsubscribe(UnsubscribeRequest) returns (UnsubscribeResponse);~~**

**~~rpc updateNotificationDelivery(NotificationDeliveryRequest) returns (NotificationDeliveryResponse);~~**

rpc checkTokenValidity(TokenValidityRequest) returns (TokenValidityResponse);

rpc synchronizeSubscriptions(SynchronizeSubsRequest) returns (SynchronizeSubsResponse);

**~~rpc updateProfileName(ChangeProfileNameRequest) returns (ChangeProfileNameResponse);~~**

|  |  |
| --- | --- |
|  |  |
| **rpc** **addNewUser(NewUserRequest)**  **~~rpc updateGCMToken(UpdateTokenRequest~~**  **~~rpc updateProfileName(ChangeProfileNameRequest) returns (ChangeProfileNameResponse);~~**  message UserAndroidInfo {  string loginToken  string gcmToken = 1;  string appName = 2;  string appVersion = 3;  string androidSecureID = 4;  string deviceFingerprint = 5;  string gsfId = 6;  string countryCode = 7;  string ipAddress = 8; }  message NewUserResponse{  string authToken = 1; } | **POST: /api/createOrUpdateAndroidUser**  Request:  //----------------------------------------------  public class UserAndroidInfo {  *private String loginToken;*  private String gcmToken;  private String appName;  private String appVersion;  private String androidSecureID;  private String deviceFingerprint;  private String gsfId;  private String countryCode;  private String ipAddress;  }  Response:  //----------------------------------------------  public class CreatedUser {  private String loginToken  } |
| **rpc addNewSubscription(NewSubscriptionRequest)**  **~~rpc unsubscribe(UnsubscribeRequest) returns (UnsubscribeResponse);~~**  **~~rpc updateNotificationDelivery(NotificationDeliveryRequest) returns (NotificationDeliveryResponse);~~**  message NewSubscriptionRequest{  string authToken = 1;  SubscriptionData subscription = 2; }  message SubscriptionData{  uint64 phoneNum = 1;  string assignedName = 2;  bool notificationsEnabled = 3; } | **POST: /api/createOrUpdateSubscription**  QueryParams:  //----------------------------------------------------  final String loginToken = QueryParam("loginToken");  Request:  //-------------------------------------------------  public class UserSubscriptionInfo {  private Long actualDate;  private String subscriptionName;  private String phoneNum;  private Byte subscriptionStatusId;  private Boolean onlineNotify;  }  subscriptionStatusId:  0 – Created  1 - Confirmed  2 - Closed  **-1 – Cancelled**  **-2 - Phone not exists**  **-3 - Banned**  //------------------------------------------------  Response:  public class CreatedUserSubscription {  private Integer substrionptionId;  private Integer answerCode;  private String note;  } |
| **rpc** **getSubscriptionNumberState(SubscriptionNumberStateRequest) returns (SubscriptionNumberStateResponse);** | **GET: /api/getSubscription**  QueryParams:  //----------------------------------------------------  final String phoneNum = QueryParam("phone");  final String loginToken = QueryParam("loginToken");  Response:  //----------------------------------------------------  public class UserSubscriptionInfo {  private Long actualDate;  private String subscriptionName;  private String phoneNum;  private String loginToken;  private Byte subscriptionStatusId;  private Boolean onlineNotify;  } |
| **rpc** **getMySubscriptions(SubscriptionsRequest) returns (SubscriptionsResponse);**  message SubscriptionsRequest{ string authToken = 1;}  message SubscriptionsResponse{ repeated SubscriptionData subscription = 1;}  message SubscriptionData{ uint64 phoneNum = 1; string assignedName = 2; bool notificationsEnabled = 3;} | **GET: /api/getAllSubscriptions**  QueryParams:  //----------------------------------------------------  final String loginToken = QueryParam("loginToken");  //=================================  Response:  private Collection<UserSubscriptionInfo> records;  public class UserSubscriptionInfo {  private Long actualDate;  private String subscriptionName;  private String phoneNum;  private Byte subscriptionStatusId;  private Boolean onlineNotify;  } |
|  | **GET: /api/getActivities**  QueryParams:  //----------------------------------------------------  final String phoneNum = QueryParam("phone");  final String loginToken = QueryParam("loginToken");  final LocalDateTime d1 =QueryParam("d1")));  final LocalDateTime d2 =QueryParam("d2")));  //=================================  Response:  private Collection<ActivityRecord> records;  public class ActivityRecord {  private Integer subscriptionId;  private LocalDateTime actualDate;  private Boolean isOnline;  } |
|  | **GET: /api/****getModifiedSubscriptions**  QueryParams:  //----------------------------------------------------  final Long actualDate =QueryParam("actualDate")));  //=================================  Response:  private Collection<UserSubscriptionInfo> records;  public class UserSubscriptionInfo {  private Long actualDate;  private String subscriptionName;  private String phoneNum;  private Byte subscriptionStatusId;  private Boolean onlineNotify;  } |
|  | **GET: /api/****getModifiedDevices**  QueryParams:  //----------------------------------------------------  final Long actualDate  //=================================  Response:  private Collection<UserDeviceShortInfo> records;  public class UserSubscriptionInfo {  private Long actualDate;  private Integer userId;  private String gcmToken;  private String secureId;  } |
| **InsertPresenceNoteBulkCall**  val sqlQuery =  "insert into visitnote (phone\_id, is\_online, add\_time) values (?, ?, cast(sys\_extract\_utc(systimestamp) as date)) LOG ERRORS REJECT LIMIT UNLIMITED"  it.prepareStatement(sqlQuery).use { ps ->  for (pair in listPhoneDBIdOnIsOnline) {  ps.setInt(1, pair.first)  ps.setBoolean(2, pair.second)  ps.addBatch()  }  ps.executeBatch()  } | // регистрация отметок активности  **POST: /api/createActivities**  private Collection<ActivityRecord> records  public class ActivityRecord {  private Integer subscriptionId;  private Long actualDate;  private Boolean isOnline;  } |
| **getAuthkeysByGroup(groupName: String)**  var sqlQuery = "select authkey.phone\_num, authkey.payload from authkey join botgroup on (authkey.group\_id=botgroup.id) where botgroup.name = ?"  it.prepareStatement(sqlQuery).use{ps ->  ps.setString(1, groupName)  ps.executeQuery().use {rs ->  generateSequence {  if (rs.next()) Pair(rs.getLong(1), rs.getString(2)) else null  }  .filterNotNull()  .map {  Authkey(it.first, it.second)  }  .toList()  }  } | // получить всех агентов с нужным статусом  **GET: /api/****getAgentsList**  QueryParams:  //----------------------------------------------------  final Byte agentStatus  AS\_RESERVE = 1  AS\_SUPPORT = 2  AS\_TRACKNG = 3  AS\_BANNED = 4  AS\_QUARANTINE = 5  //==================================  private Collection<AgentInfo> agents  public class AgentInfo {  private Integer agent\_id;  private String phone\_num;  private String payload;  private Long actual\_date;  private Byte agent\_status\_id;  } |
| **fun** **replaceBannedAuthkey(phoneNum: Long, botType: BotType): String?=**  try {  OracleHikariDataSource.dataSource.connection.use {  val isTracker = botType == BotType.TRACKER  it.prepareCall("begin ? := wa\_tracking.ban\_bot(?,?); end;").use { cstmt ->  cstmt.registerOutParameter(1, Types.VARCHAR)  cstmt.setLong(2, phoneNum)  cstmt.setBoolean(3, isTracker)  cstmt.execute()  cstmt.getString(1)  }  }  }  catch(ex: Exception){  DBManager.log.error("error replacing banned authkey", ex)  null  } | // изменить статус агента/бота  **GET /api/updateAgentStatus**  QueryParams:  //----------------------------------------------------  final String phoneNum  final Byte agentStatus  AS\_RESERVE = 1  AS\_SUPPORT = 2  AS\_TRACKNG = 3  AS\_BANNED = 4  AS\_QUARANTINE = 5  //================================  public class AgentPayloadInfo {  private String authKey;  }  // если новый статус агента = (4,5), то на подписках агента заменяется агент (подбирается наименее загруженный) |
| **fun replaceQuarantinedAuthkey(phoneNum: Long, botType: BotType): String?=**  **try {**  OracleHikariDataSource.dataSource.connection.use {  val isTracker = botType == BotType.TRACKER  it.prepareCall("begin ? := wa\_tracking.quarantine\_bot(?,?); end;").use { cstmt ->  cstmt.registerOutParameter(1, Types.VARCHAR)  cstmt.setLong(2, phoneNum)  cstmt.setBoolean(3, isTracker)  cstmt.execute()  cstmt.getString(1)  }  }  }  catch(ex: Exception){  DBManager.log.error("error replacing quarantined authkey", ex)  null  **}** | см. предыдущий пункт |
| **fun getSubscriptionNumbersForBot(phoneNum: Long, considerBotFreshHours: Int) :AuthkeyConfig?=**  try {  OracleHikariDataSource.dataSource.connection.use {  var sqlQuery =  """select authkey.phone\_num,  | subscriptionphone.phone\_num,  | subscriptionphone.id,  | case when cast(sys\_extract\_utc(systimestamp) as date)-subscriptionphone.add\_time<?/24 then 1 else 0 end as isFresh,  | case when cast(sys\_extract\_utc(systimestamp) as date)-authkey.last\_change\_time<?/24 then 1 else 0 end as botIsFresh  | from authkey join subscriptionphone on (subscriptionphone.key\_id=authkey.id) where authkey.phone\_num = ?""".trimMargin()  it.prepareStatement(sqlQuery).use { ps ->  ps.setInt(1,considerBotFreshHours)  ps.setInt(2,considerBotFreshHours)  ps.setLong(3, phoneNum)  ps.executeQuery().use { rs ->  generateSequence {  if (rs.next()) Quadruple(rs.getLong(1), rs.getLong(2), rs.getInt(3), rs.getBoolean(4) or rs.getBoolean(5)) else null  }  .filterNotNull()  .groupBy { it.first }  .mapValues {  ConcurrentHashMap<Long, Pair<Int,Boolean>>().apply {  for (listItem in it.value)  put(listItem.second, Pair(listItem.third,listItem.fourth))  }  }  .map {  AuthkeyConfig(it.key, it.value)  }  .firstOrNull()  }  }  }  } | // получить список подписок агента\бота  **GET /api/getAgentSubscriptions**  QueryParams:  final String agentPhoneNum (agentPhone)  ~~final String freshHours; (freshHours)~~  //=================================  public class UserSubscriptionInfoCollection {  private Collection<UserSubscriptionInfo> userSubscriptionInfoCollection;  }  public class UserSubscriptionInfo {  private Long actualDate;  private String subscriptionName;  private String phoneNum;  private Byte subscriptionStatusId;  private Boolean onlineNotify;  } |
| **fun updateSubscriptionNumberState(subscriptionPhoneId: Int, isValid: Boolean) =**  try {  OracleHikariDataSource.dataSource.connection.use {  it.prepareCall("call wa\_tracking.update\_subscription\_state(?,?)").use { cstmt ->  cstmt.setInt(1, subscriptionPhoneId)  cstmt.setBoolean(2, isValid)  cstmt.executeUpdate()  }  }  }  catch(ex: Exception){  DBManager.log.error("error updating subscriptions state", ex)  } | // изменить статус подписки  **POST /api/updateSubscriptionStatus**  QueryParams:  //==================================  final Integer subscriptionId;  final String stringSubscriptionStatus;  SS\_CREATED = 0  SS\_CONFIRMED = 1  SS\_CLOSED = 2  SS\_CANCELLED = -1  SS\_PHONE\_NOT\_EXISTS -2  SS\_BANNED = -3  //================================== |
|  | // получить список пользовательских устройств (на тек. момент андроид) , у которых изменились атрибуты  **GET: /api/getModifiedDevices**  //==================================  private Collection<UserDeviceShortInfo> userDevices  public class UserDeviceShortInfo {  private Long actualDate;  private Integer userId;  private String gcmToken;  private String secureId;  } |
|  |  |

**+** **insertPresenceNoteBulkCall**

**+ getAuthkeysByGroup**

**+ replaceBannedAuthkey**

**+ replaceQuarantinedAuthkey**

**+ getSubscriptionNumbersForBot**

**++++ updateSubscriptionNumberState**

+ updateAvatar

--- getNewSubscriptionData

+ getNotificationMap

--- getNotifDeliveryStatus

--- getNotifDeliveryStatusForUser

--- getUpdatedSubscriptionActiveState

??? getUnprocessedPhonesCall